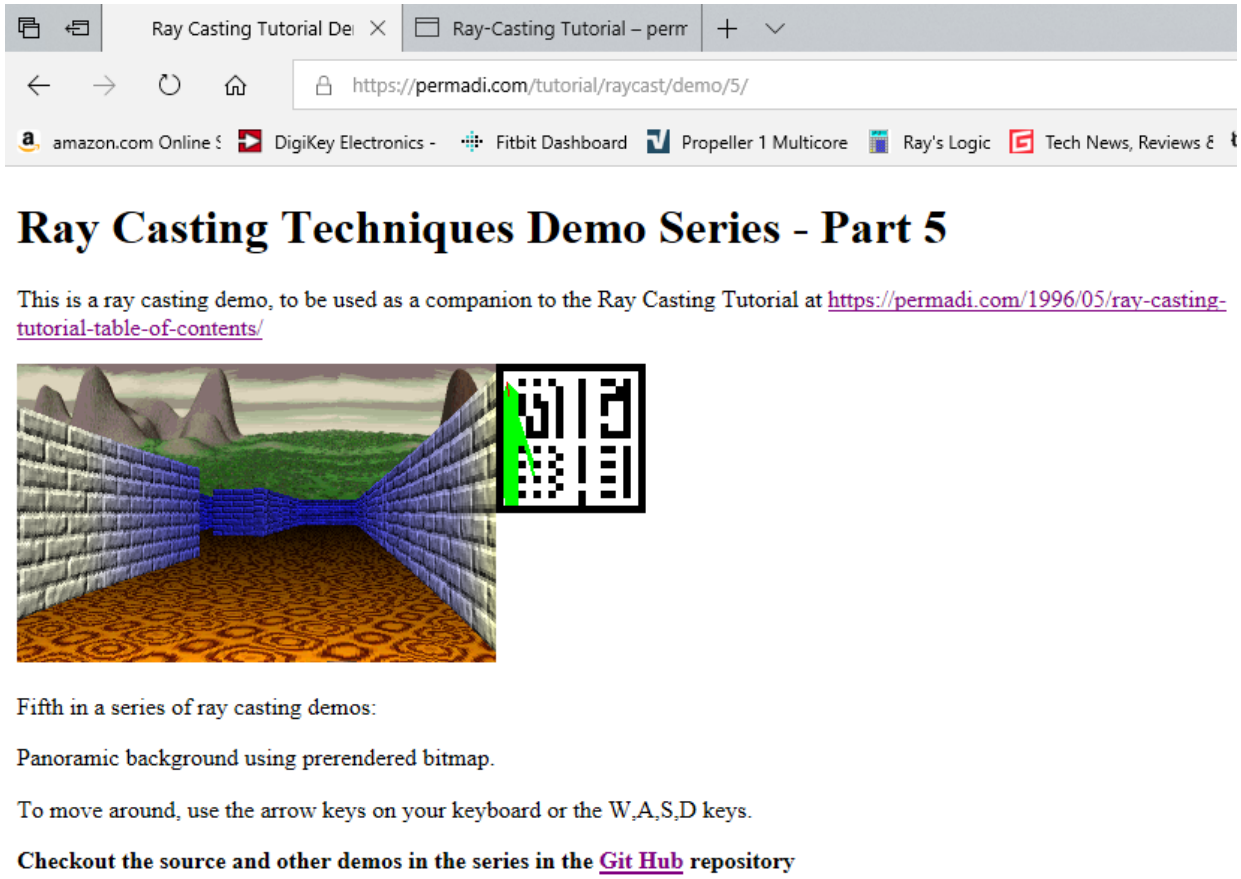
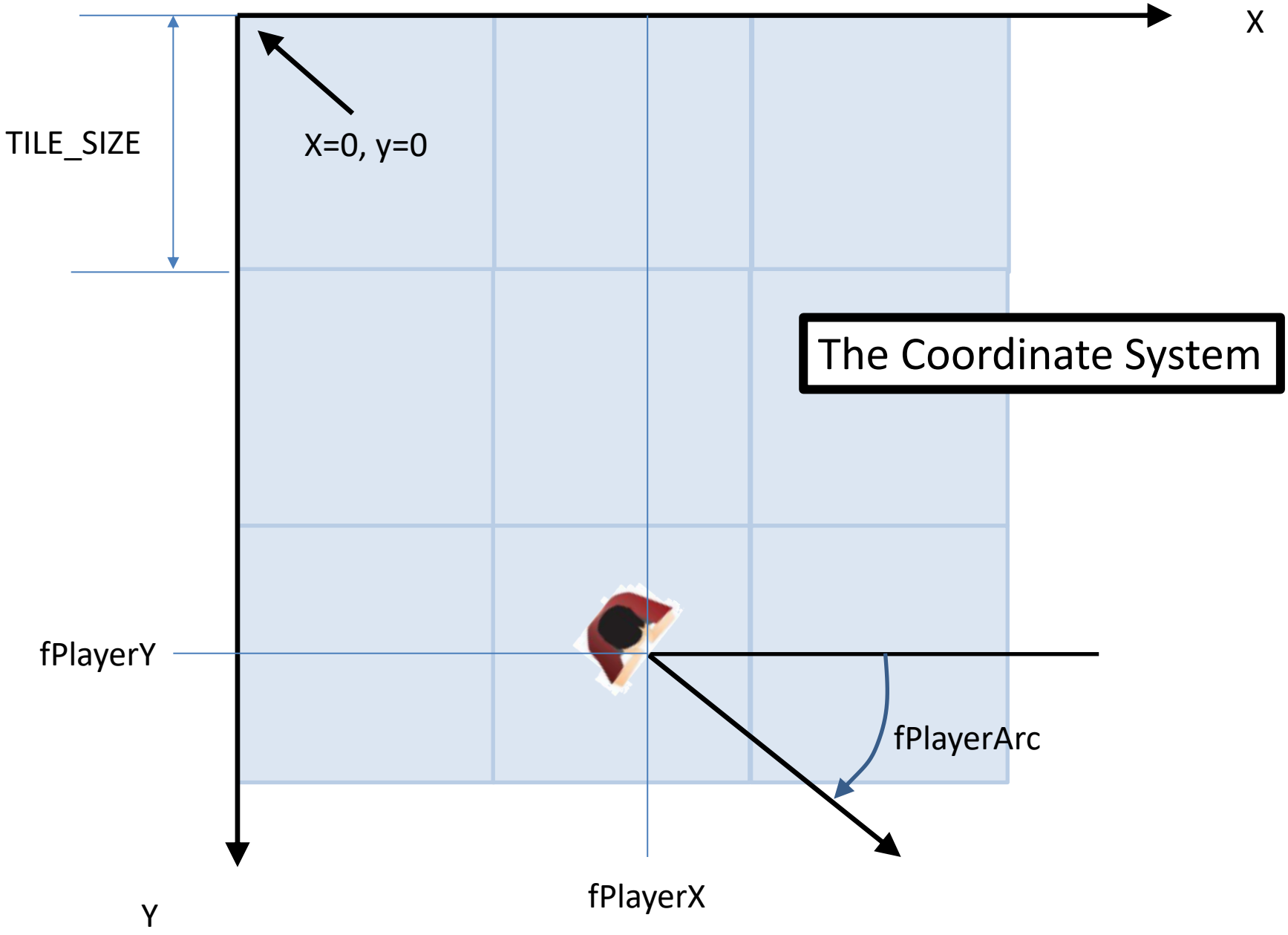
# Raycasting Notes

RJA Dec18

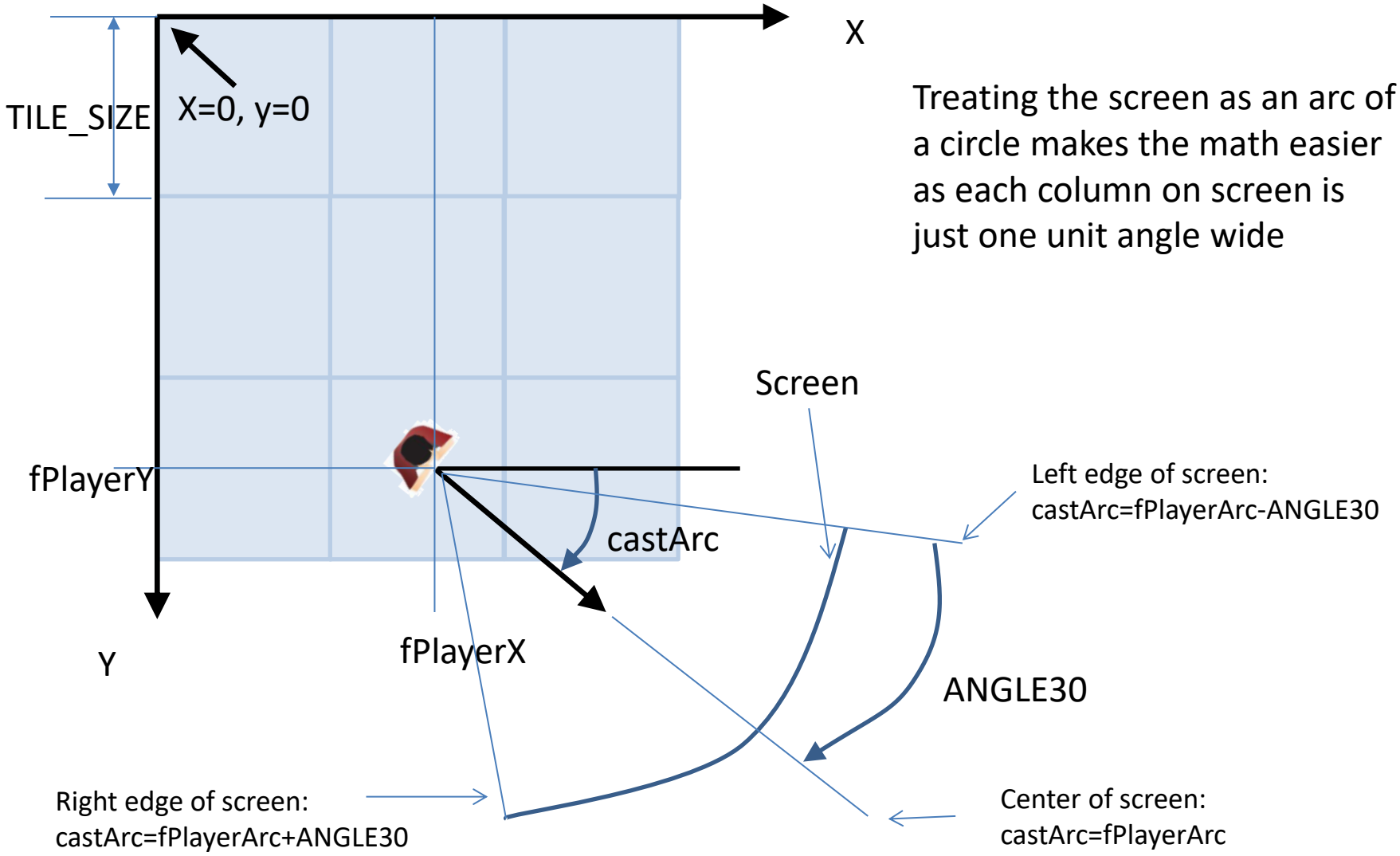
# The permadi.com raycasting Tutorial is great, but need more visualization to understand math




Note: Be sure to read the tutorial: https://permadi.com/1996/05/ray-casting-tutorial-table-of-contents/

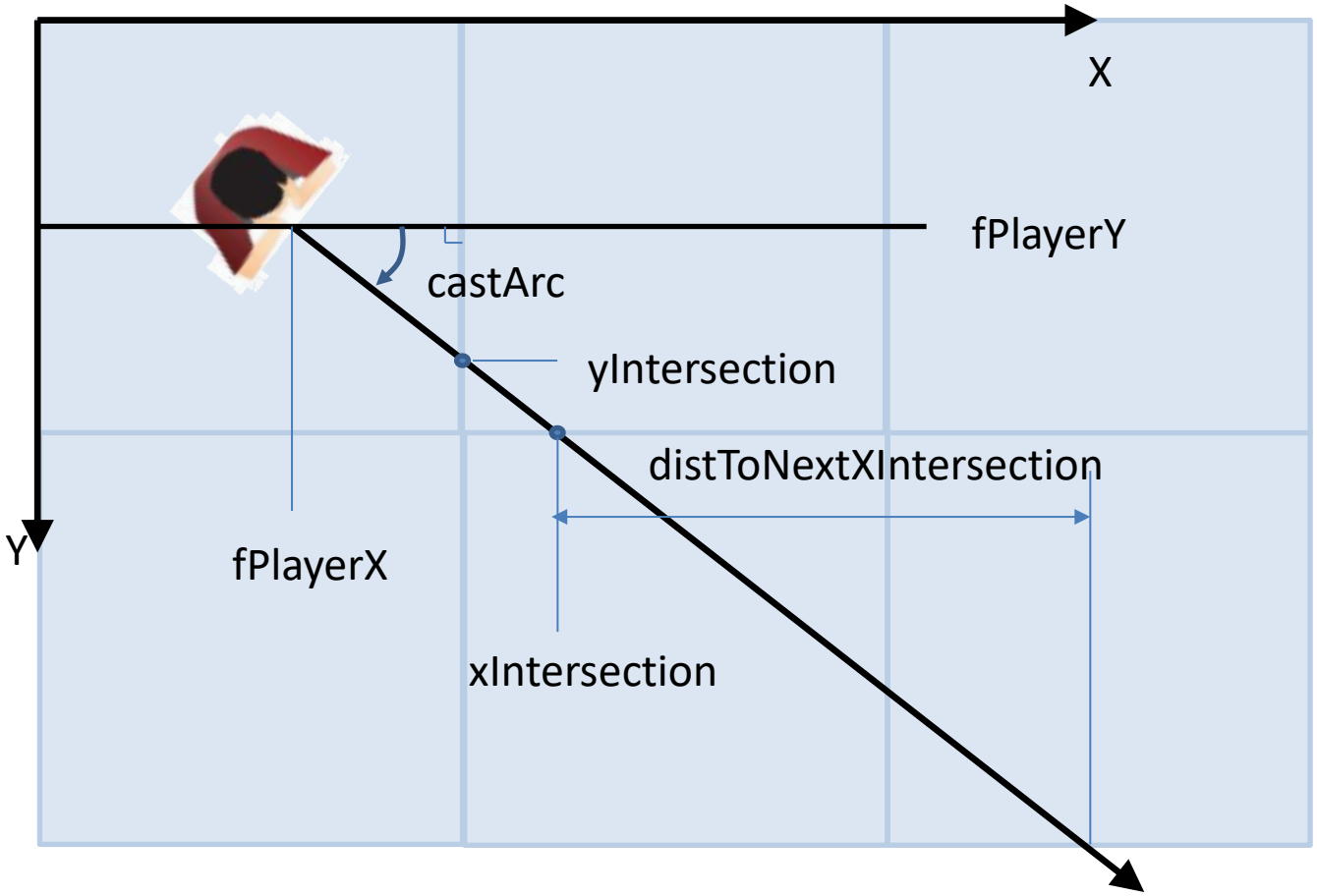



Note: Y axis goes down instead of the usual up. Same way screen data stored memory.



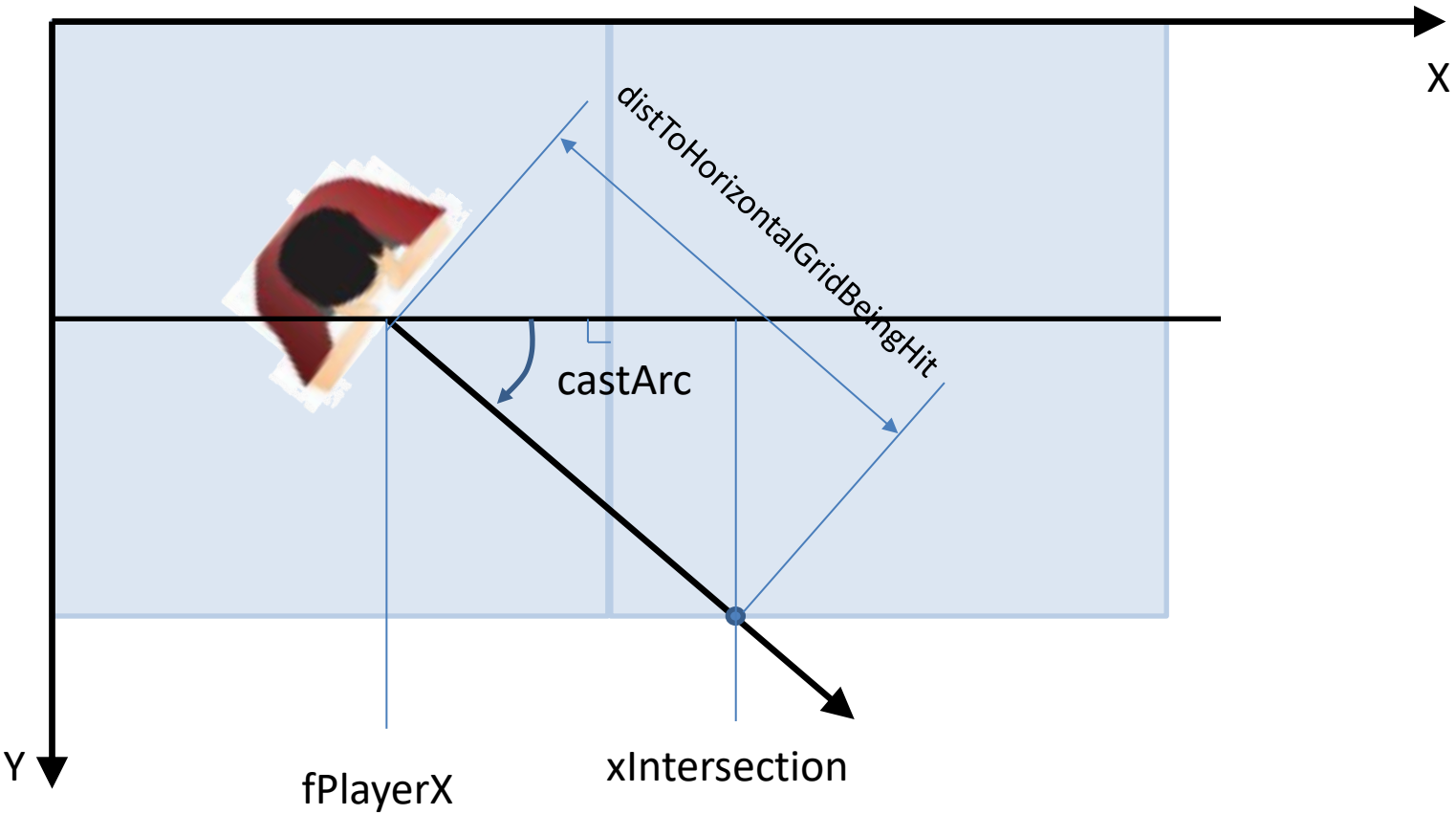

- The projections screen represents a 60 degree field of view, maps directly to PROJECTIONPLANEWIDTH
- Each column on screen represents one angular unit from fPlayerArc-ANGLE30 to fPlayerArc+ANGLE30
- Initialize: castArc=fPlayerArc-ANGLE30
- Note that ANGLE60 is equal to number of columns on screen (320 in this example).
- The main loop is over columns on the screen (from 0 to 319), iteration variable is castColumn
- At end of loop, we do castArc++ to increment angle
- Note: Pretending screen is curved causes a “fish bowl” distortion that is corrected for before drawing to screen

For each castArc, we find first intersection of our cast ray with vertical and horizontal walls




- If the first intersections we find are an opening on map and not a wall, just add a fixed number (from a table) to get next intersection:
  - distToNextXIntersection = this.fXStepTable[castArc];
  - xIntersection += distToNextXIntersection;
- We repeat this in a loop until we find both x and y nearest walls

We use 1/cos table (fICosTable)to calculate distance from player to xIntersection

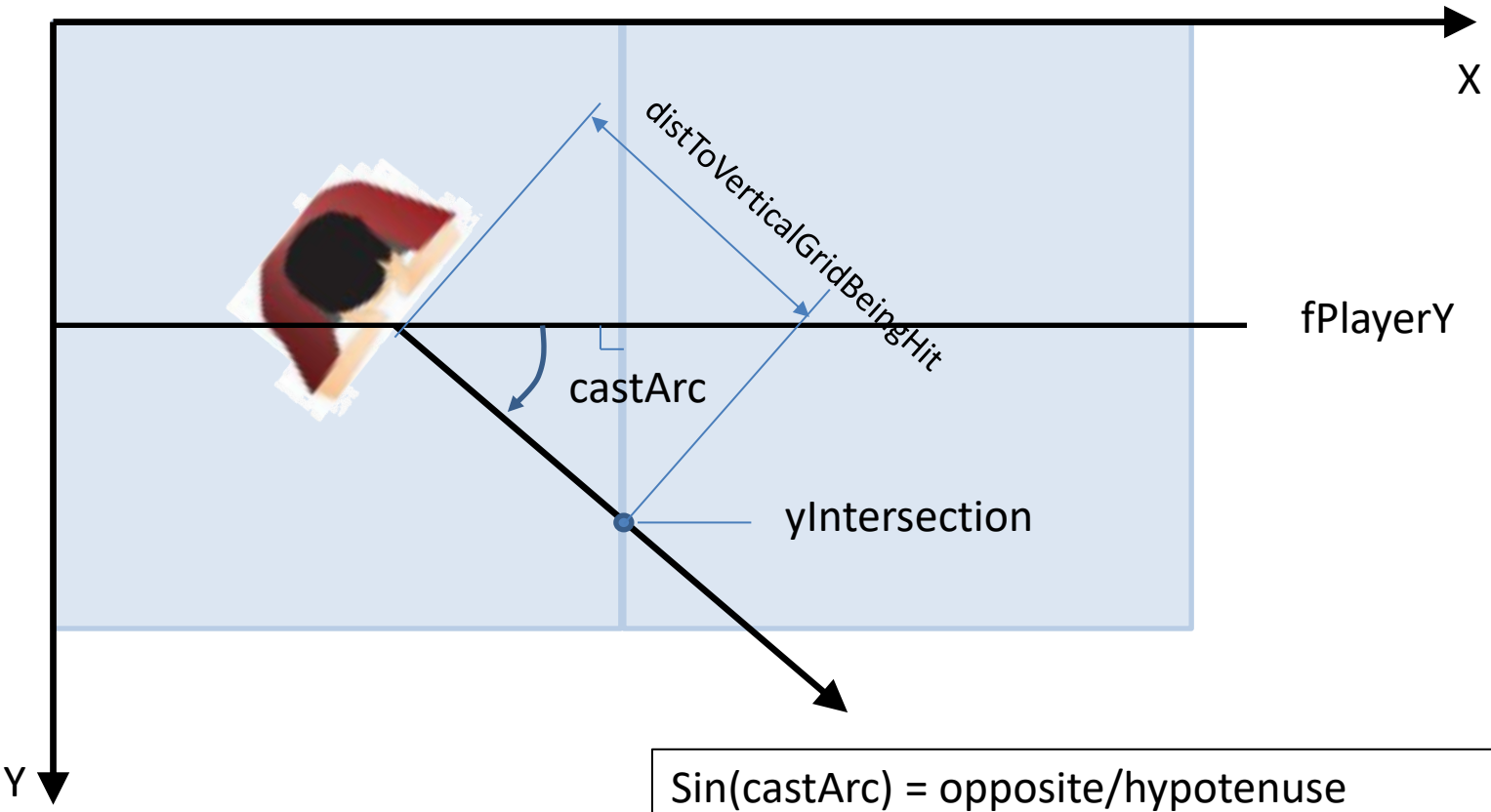



Cos(castArc) = adjacent/hypotenuse
Adjacent= xIntersection-fPlayerX
Hypotenuse= distToHorizontalGridBeingHit
distToHorizontalGridBeingHit=adjacent/Cos(castArc)

distToHorizontalGridBeingHit = (xIntersection - this.fPlayerX)* this.fICosTable[castArc];

ICos=1/cos

Similarly, use 1/sin table to calculate distance from player to yIntersection

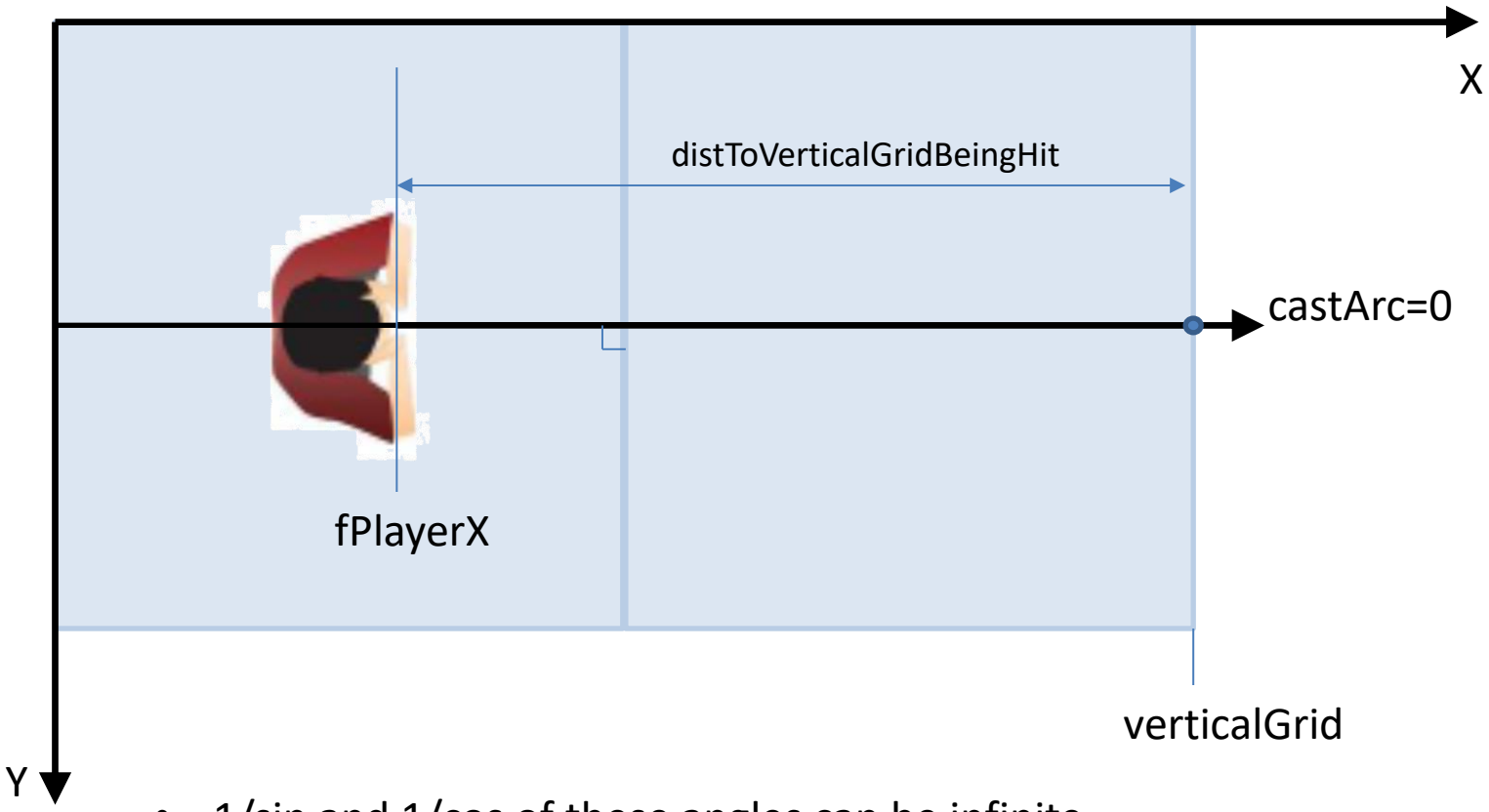



Sin(castArc) = opposite/hypotenuse
Opposite= yIntersection-fPlayerY
Hypotenuse= distToVerticalGridBeingHit
distToVerticalGridBeingHit=opposite/Sin(castArc)

distToVerticalGridBeingHit =(yIntersection- this.fPlayerY)* this.fISinTable[castArc];

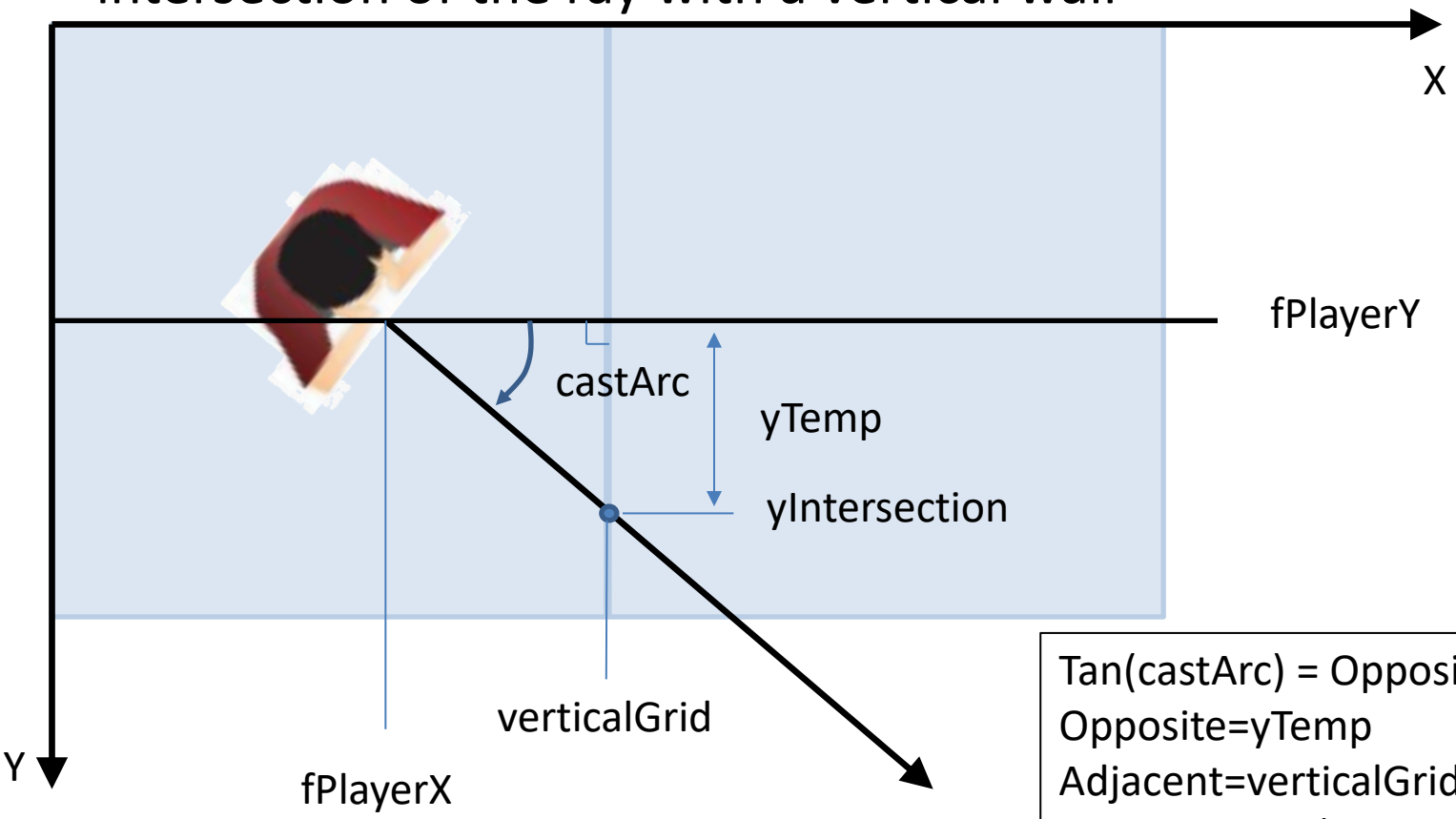
ISin=1/sin

Need to handle angles 0, 90, 180 and 270 with care




- 1/sin and 1/cos of these angles can be infinite
- The javascript example adds a hair to each angle in table
  - Trig value comes out very large instead of infinite
- Large value times tiny value of (yIntersection-fPlayery) turns out OK
- Possibly a better way is to simply use:
  - Angle=0:  distToVerticalGridBeingHit = verticalGrid – fPlayerX
  - Angle=180:  distToVerticalGridBeingHit = fPlayerX – verticalGrid

# Finding **yIntersection facing right**, the y-coordinate of first intersection of the ray with a vertical wall

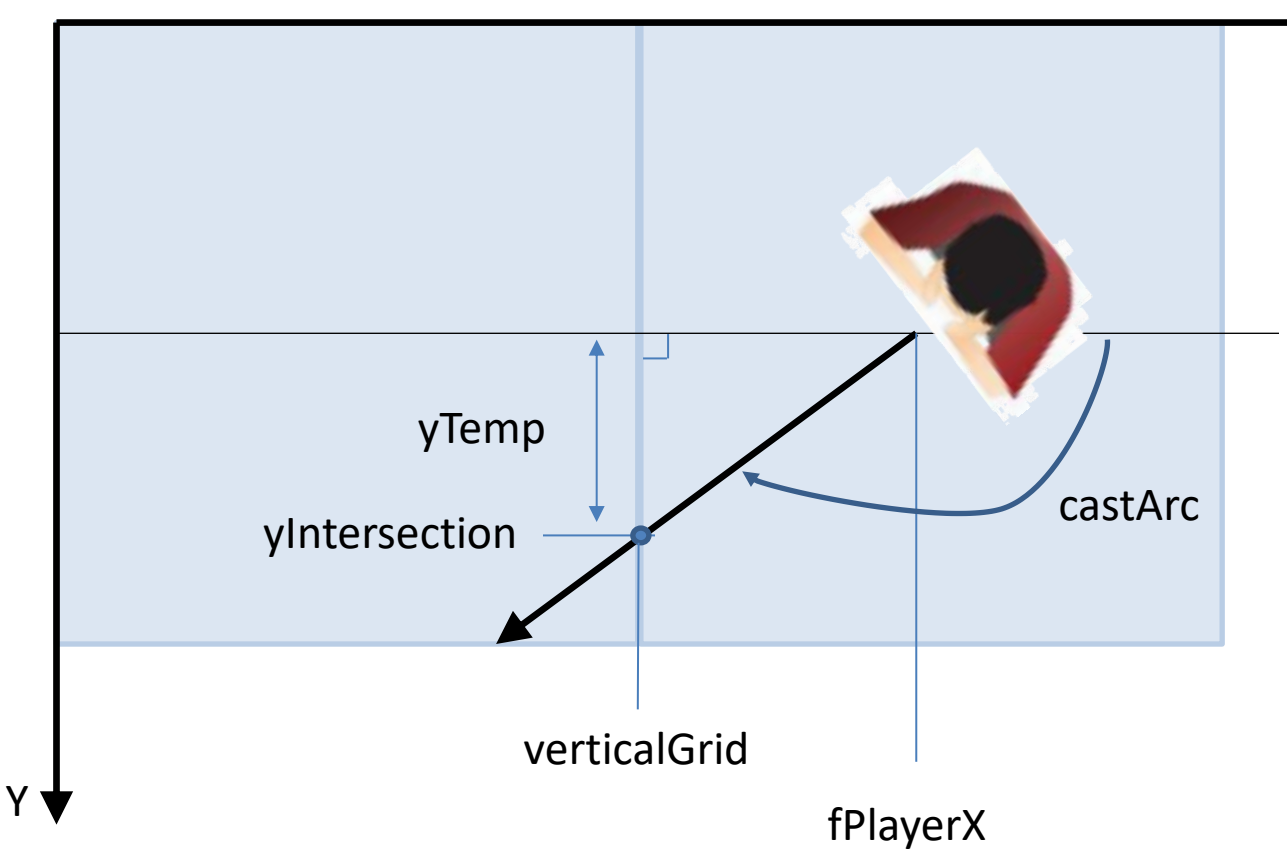


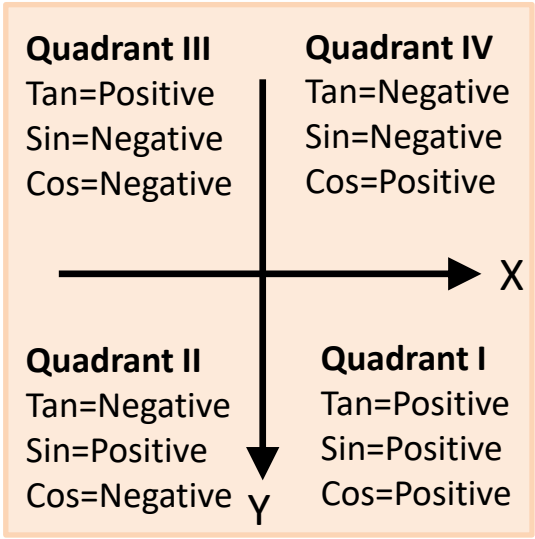

Tan(castArc) = Opposite/Adjacent
Opposite=yTemp
Adjacent=verticalGrid-fPlayerX
Opposite=Tan(castArc)*Adjacent

For ray **Facing Right** (castArc<ANGLE90 ) OR (castArc>ANGLE270):

verticalGrid=TILE_SIZE + floor( fPlayerX / TILE_SIZE ) * TILE_SIZE
yTemp = Tan(castArc) * (verticalGrid - fPlayerX)
yIntersection= fPlayerY + yTemp

**When Facing Left (Quadrants II and III):** verticalGrid is TILE_SIZE smaller



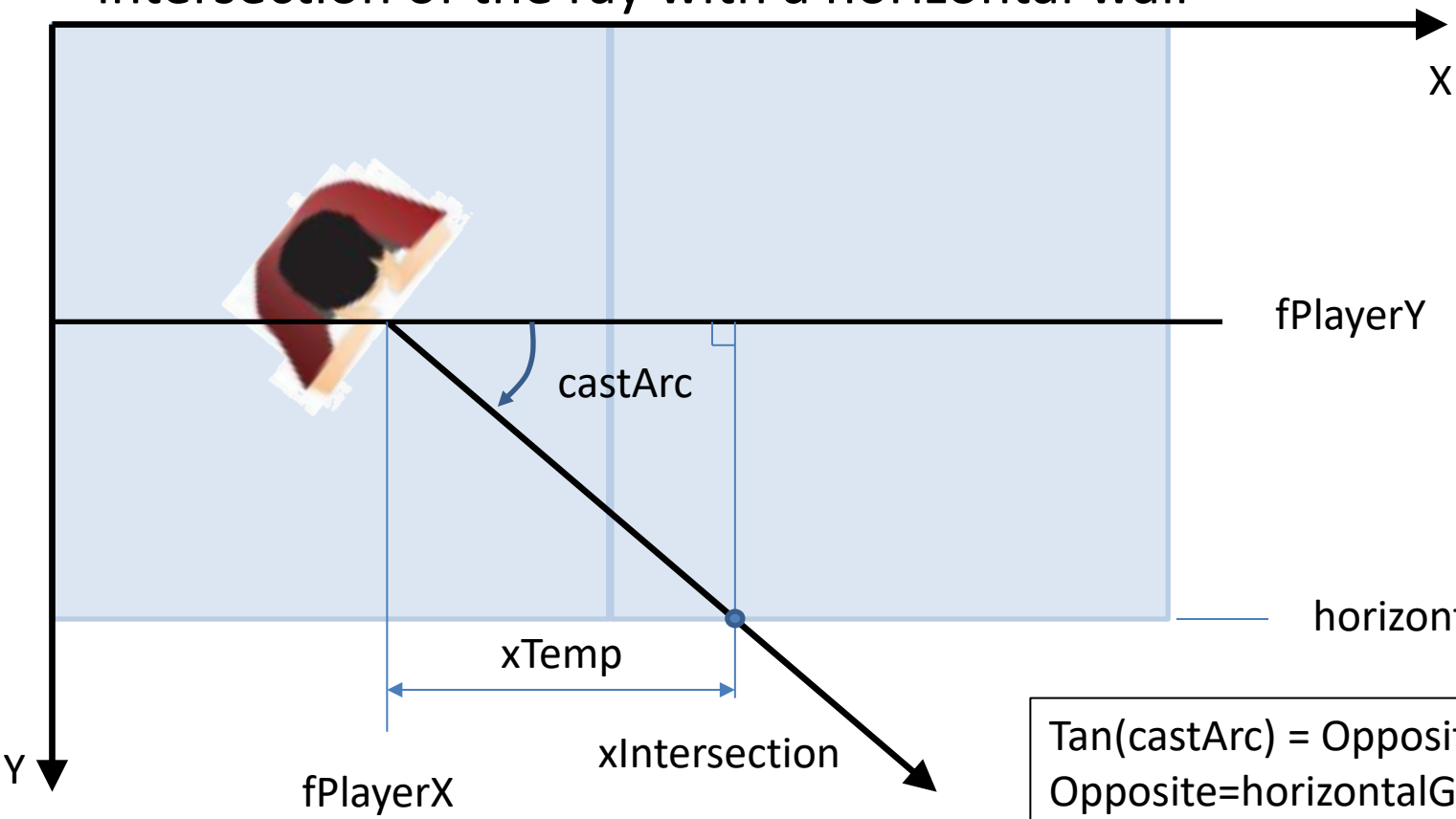




For **ray facing left** (castArc>ANGLE90 ) AND (castArc<ANGLE270):

verticalGrid= floor( fPlayerX / TILE_SIZE ) * TILE_SIZE
yTemp = Tan(castArc) * (verticalGrid - fPlayerX)
yIntersection= fPlayerY + yTemp

Negative*negative in Quadrant II

# Finding **xIntersection facing down**, the x-coordinate of first intersection of the ray with a horizontal wall



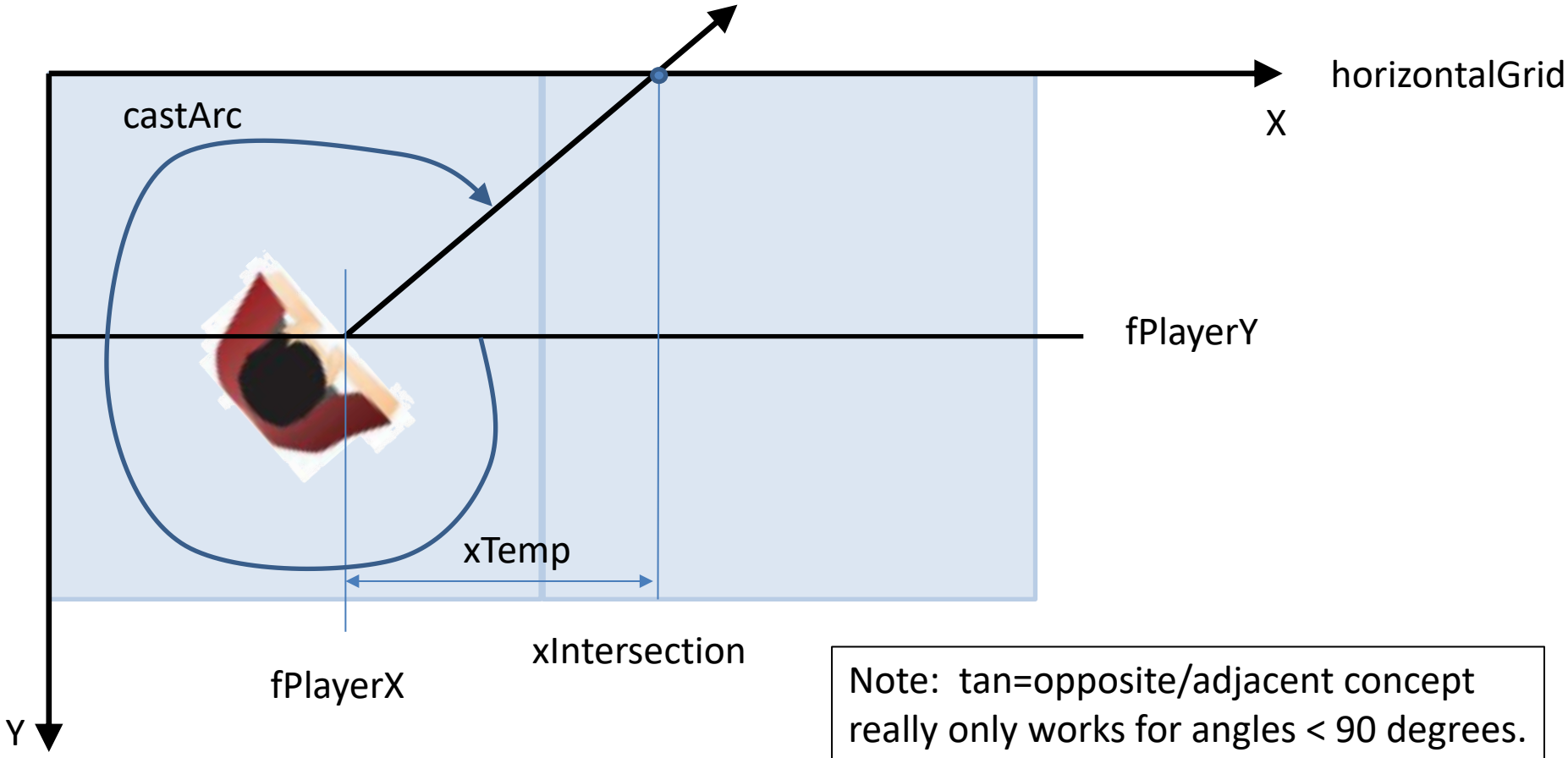

Tan(castArc) = Opposite/Adjacent
Opposite=horizontalGrid-fPlayerY
Adjacent=xTemp
Adjacent=Opposite/Tan(castArc)

For ray **Facing Down** (castArc>ANGLE0 ) AND (castArc<ANGLE180):

horizontalGrid=TILE_SIZE + floor( fPlayerY / TILE_SIZE ) * TILE_SIZE
xTemp = (horizontalGrid - fPlayerY)/ Tan(castArc)
xIntersection= fPlayerX + xTemp

Note: Code actually multiplies by 1/Tan table (fITanTable)

# Finding **xIntersection facing up**, the x-coordinate of first intersection of the ray with a horizontal wall




Note: tan=opposite/adjacent concept really only works for angles < 90 degrees. But, math still works in similar way.

For ray **Facing UP**:

horizontalGrid= floor( fPlayerY / TILE_SIZE ) * TILE_SIZE
xTemp = (horizontalGrid - fPlayerY) /Tan(castArc)
xIntersection= fPlayerX + XTemp

Note: Code actually multiplies by 1/Tan table (fITanTable)